**HOLIDAY ASSIGNMENT**

**1)PALINDROME NUMBER**

#include <stdbool.h>

bool isPalindrome(int x) {

    if (x < 0 || (x % 10 == 0 && x != 0)) return false;

    int rev = 0, original = x;

    while (x > rev) {

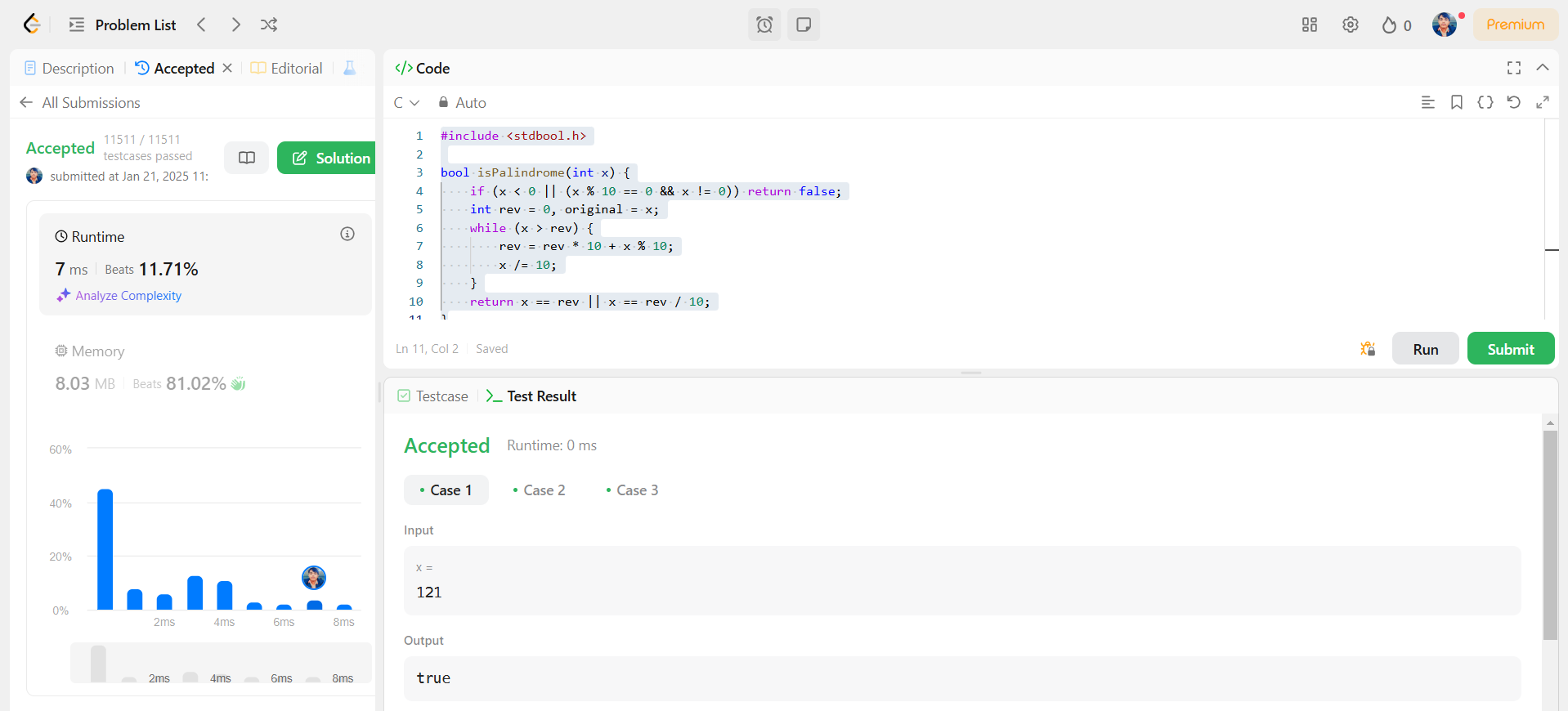
        rev = rev \* 10 + x % 10;

        x /= 10;

    }

    return x == rev || x == rev / 10;

}



**2)ROMAN TO INTEGER**

int romanToInt(char\* s) {

    int res = 0, prev = 0, curr = 0;

    while (\*s) {

        switch (\*s++) {

            case 'I': curr = 1; break;

            case 'V': curr = 5; break;

            case 'X': curr = 10; break;

            case 'L': curr = 50; break;

            case 'C': curr = 100; break;

            case 'D': curr = 500; break;

            case 'M': curr = 1000; break;

        }

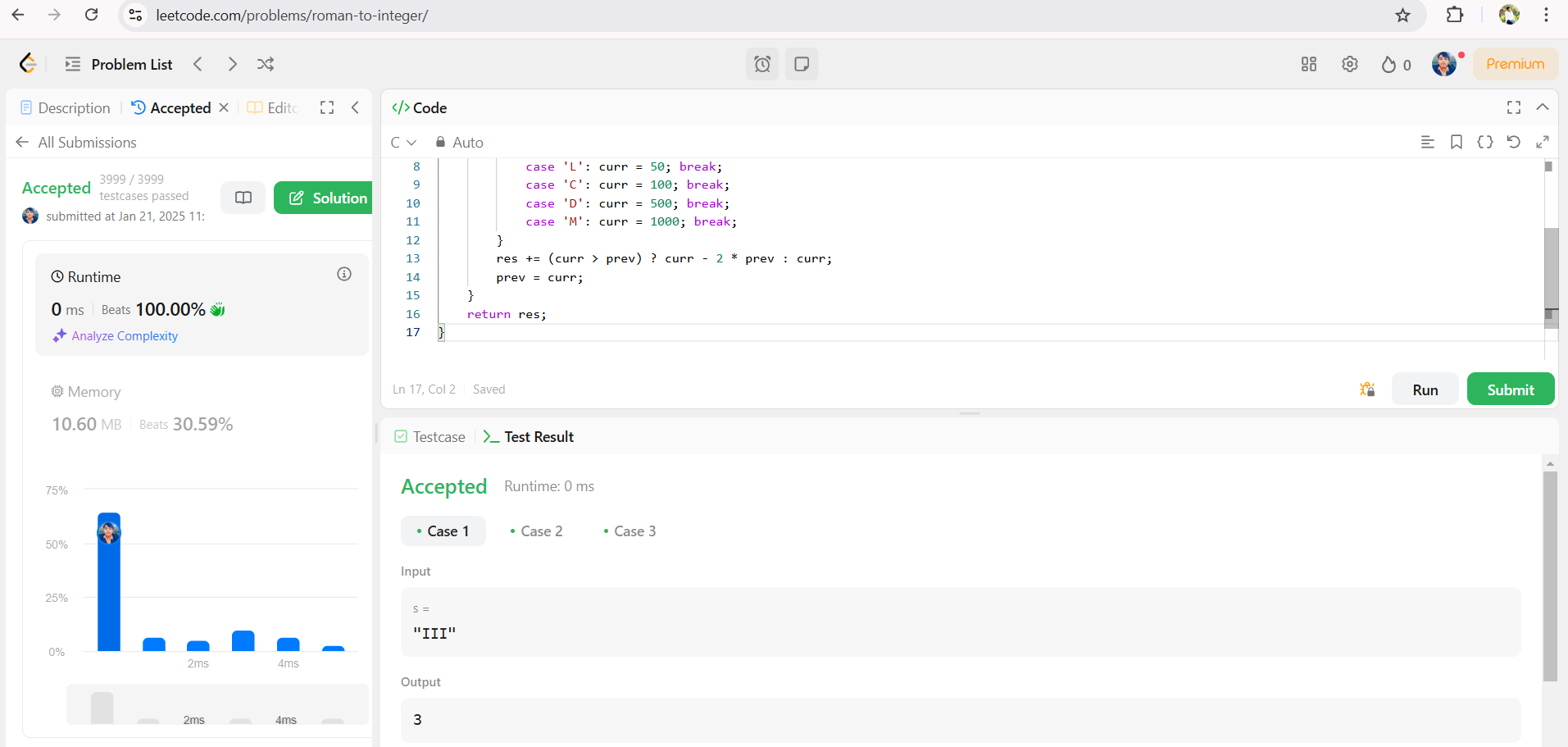
        res += (curr > prev) ? curr - 2 \* prev : curr;

        prev = curr;

    }

    return res;

}



**3)VALIDATING OPENING AND CLOSING PARENTHESIS IN A STRING**

bool canBeValid(char\* s, char\* locked) {

    int n = strlen(s), open = 0, balance = 0;

    if (n % 2 != 0) return false;

    for (int i = 0; i < n; i++) {

        if (locked[i] == '0' || s[i] == '(') open++;

        else open--;

        if (open < 0) return false;

    }

    open = 0;

    for (int i = n - 1; i >= 0; i--) {

        if (locked[i] == '0' || s[i] == ')') open++;

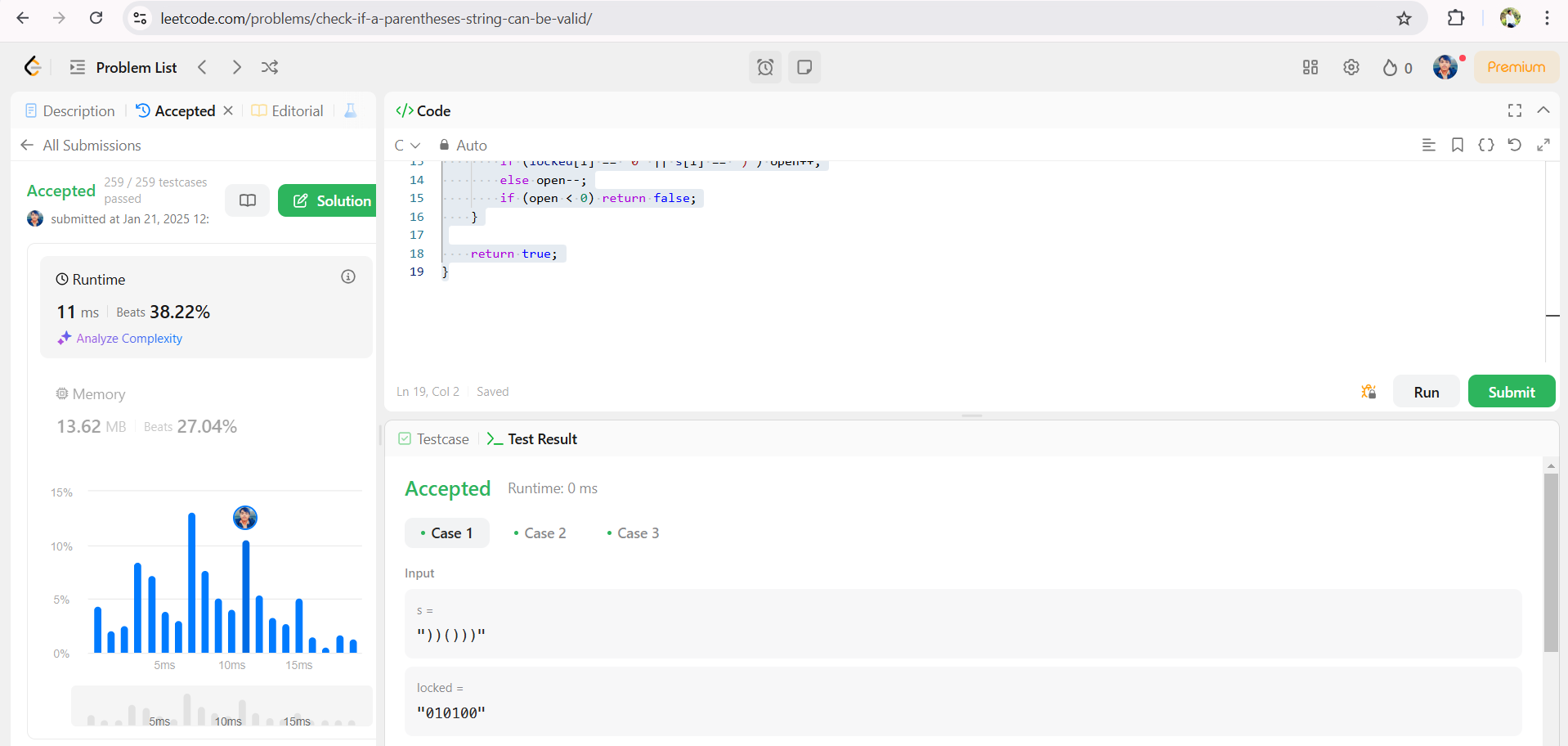
        else open--;

        if (open < 0) return false;

    }

    return true;

}



4)**FINDING ODD AND EVEN NUMBERS IN ARRAY**

#include <stdlib.h>

int\* sortArrayByParity(int\* nums, int numsSize, int\* returnSize) {

    int\* result = (int\*)malloc(numsSize \* sizeof(int));

    int start = 0, end = numsSize - 1;

    for (int i = 0; i < numsSize; i++) {

        if (nums[i] % 2 == 0) {

            result[start++] = nums[i];

        } else {

            result[end--] = nums[i];

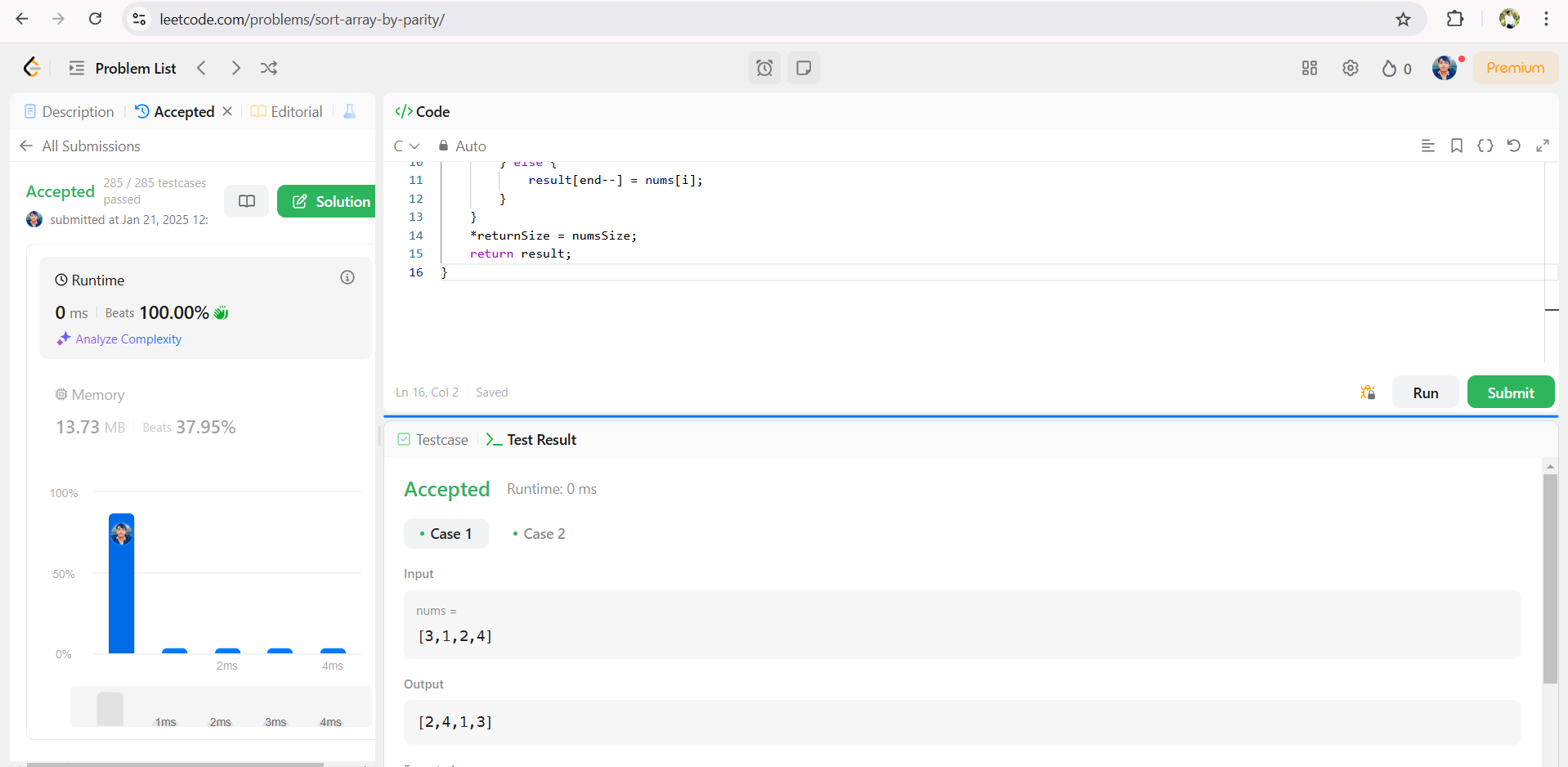
        }

    }

    \*returnSize = numsSize;

    return result;

}



**5)SYMMETRIC PAIRS OF ARRAY**

#include <stdlib.h>

int cmp(const void\* a, const void\* b) {

    return (\*(int\*)a - \*(int\*)b);

}

int findPairs(int\* nums, int numsSize, int k) {

    qsort(nums, numsSize, sizeof(int), cmp);

    int count = 0, left = 0, right = 1;

    while (right < numsSize) {

        if (left == right || nums[right] - nums[left] < k) {

            right++;

        } else if (nums[right] - nums[left] > k) {

            left++;

        } else {

            count++;

            left++;

            right++;

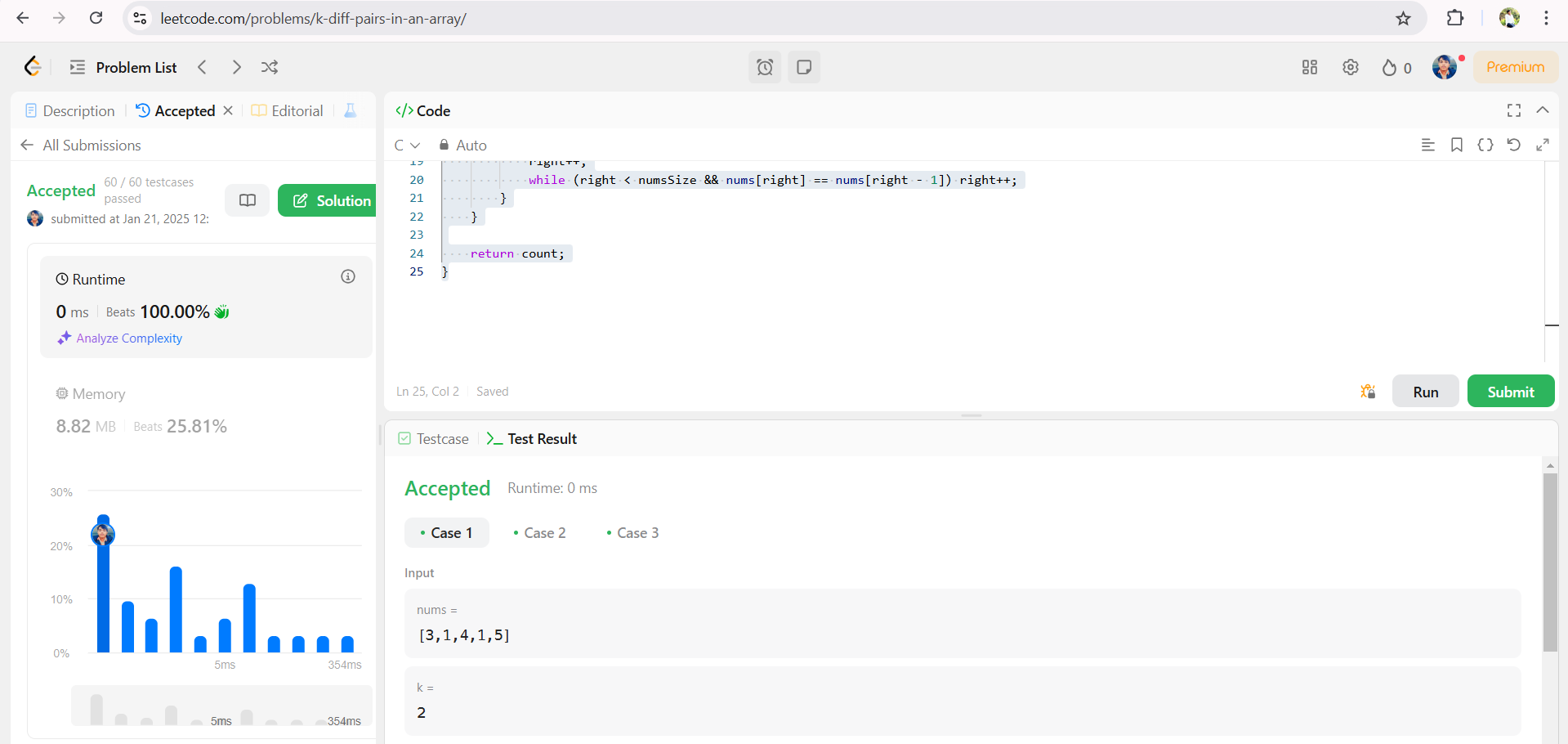
            while (right < numsSize && nums[right] == nums[right - 1]) right++;

        }

    }

    return count;

}



**6)Kth SMALLEST ELEMENT IN ARRAY**

#include <stdio.h>

#include <stdlib.h>

struct MinHeap {

    int\* arr;

    int size;

    int capacity;

};

void swap(int\* a, int\* b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void heapify(struct MinHeap\* heap, int idx) {

    int smallest = idx;

    int left = 2 \* idx + 1;

    int right = 2 \* idx + 2;

    if (left < heap->size && heap->arr[left] < heap->arr[smallest]) {

        smallest = left;

    }

    if (right < heap->size && heap->arr[right] < heap->arr[smallest]) {

        smallest = right;

    }

    if (smallest != idx) {

        swap(&heap->arr[smallest], &heap->arr[idx]);

        heapify(heap, smallest);

    }

}

void insertMinHeap(struct MinHeap\* heap, int val) {

    if (heap->size < heap->capacity) {

        heap->arr[heap->size] = val;

        heap->size++;

        int i = heap->size - 1;

        while (i > 0 && heap->arr[(i - 1) / 2] > heap->arr[i]) {

            swap(&heap->arr[(i - 1) / 2], &heap->arr[i]);

            i = (i - 1) / 2;

        }

    } else if (val > heap->arr[0]) {

        heap->arr[0] = val;

        heapify(heap, 0);

    }

}

int findKthLargest(int\* nums, int numsSize, int k) {

    struct MinHeap heap;

    heap.arr = (int\*)malloc(k \* sizeof(int));

    heap.size = 0;

    heap.capacity = k;

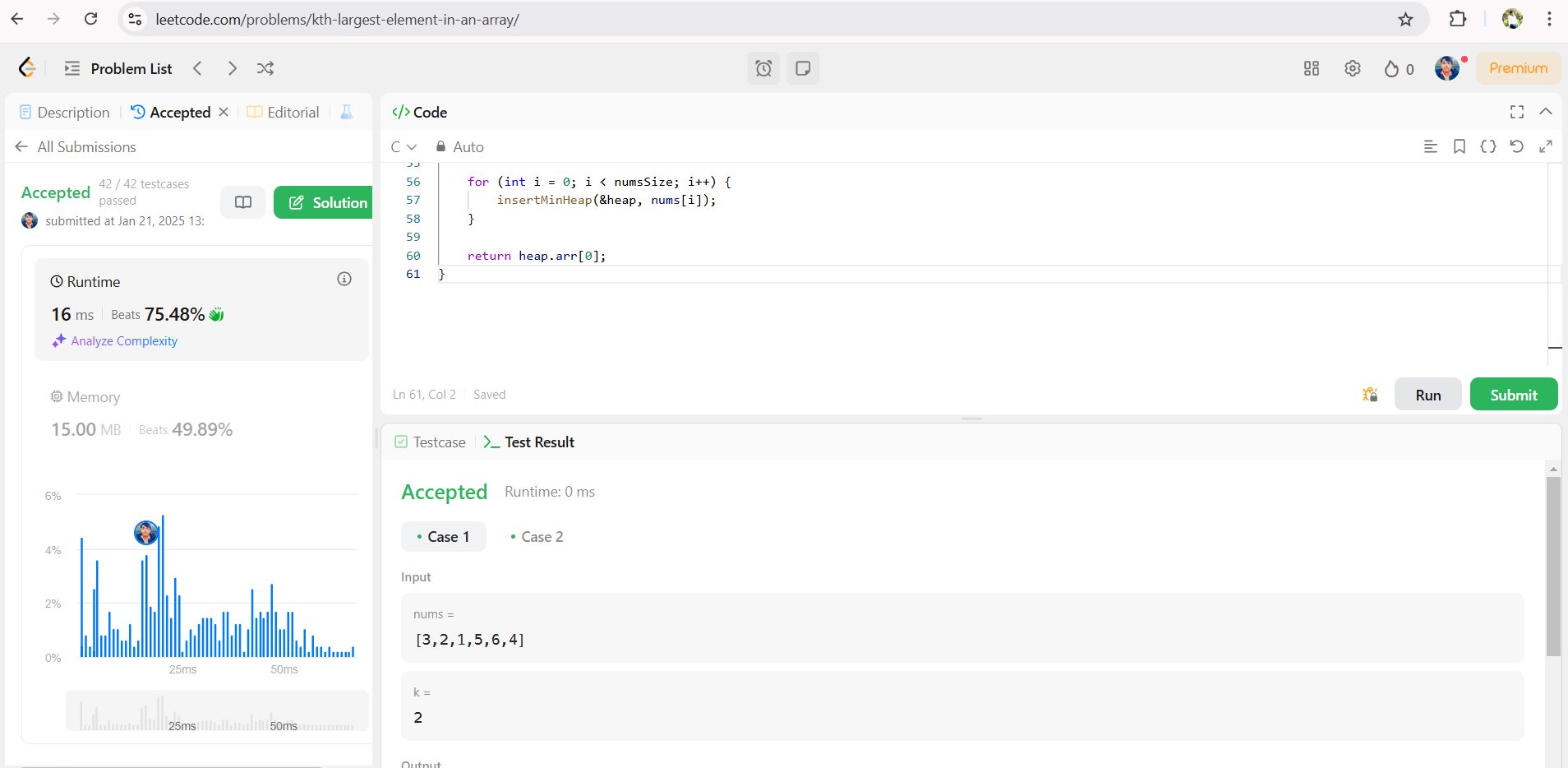
    for (int i = 0; i < numsSize; i++) {

        insertMinHeap(&heap, nums[i]);

    }

    return heap.arr[0];

}

****

**7)REPRESENT COMPLEX NUMBERS WITH REAL AND IMAGINARY PARTS**

#include <stdio.h>

#include <stdlib.h>

int\* plusOne(int\* digits, int digitsSize, int\* returnSize) {

    int carry = 1;

    for (int i = digitsSize - 1; i >= 0; i--) {

        digits[i] += carry;

        if (digits[i] == 10) {

            digits[i] = 0;

            carry = 1;

        } else {

            carry = 0;

            break;

        }

    }

    if (carry) {

        \*returnSize = digitsSize + 1;

        int\* result = (int\*)malloc(sizeof(int) \* (\*returnSize));

        result[0] = 1;

        for (int i = 1; i < \*returnSize; i++) {

            result[i] = digits[i - 1];

        }

        return result;

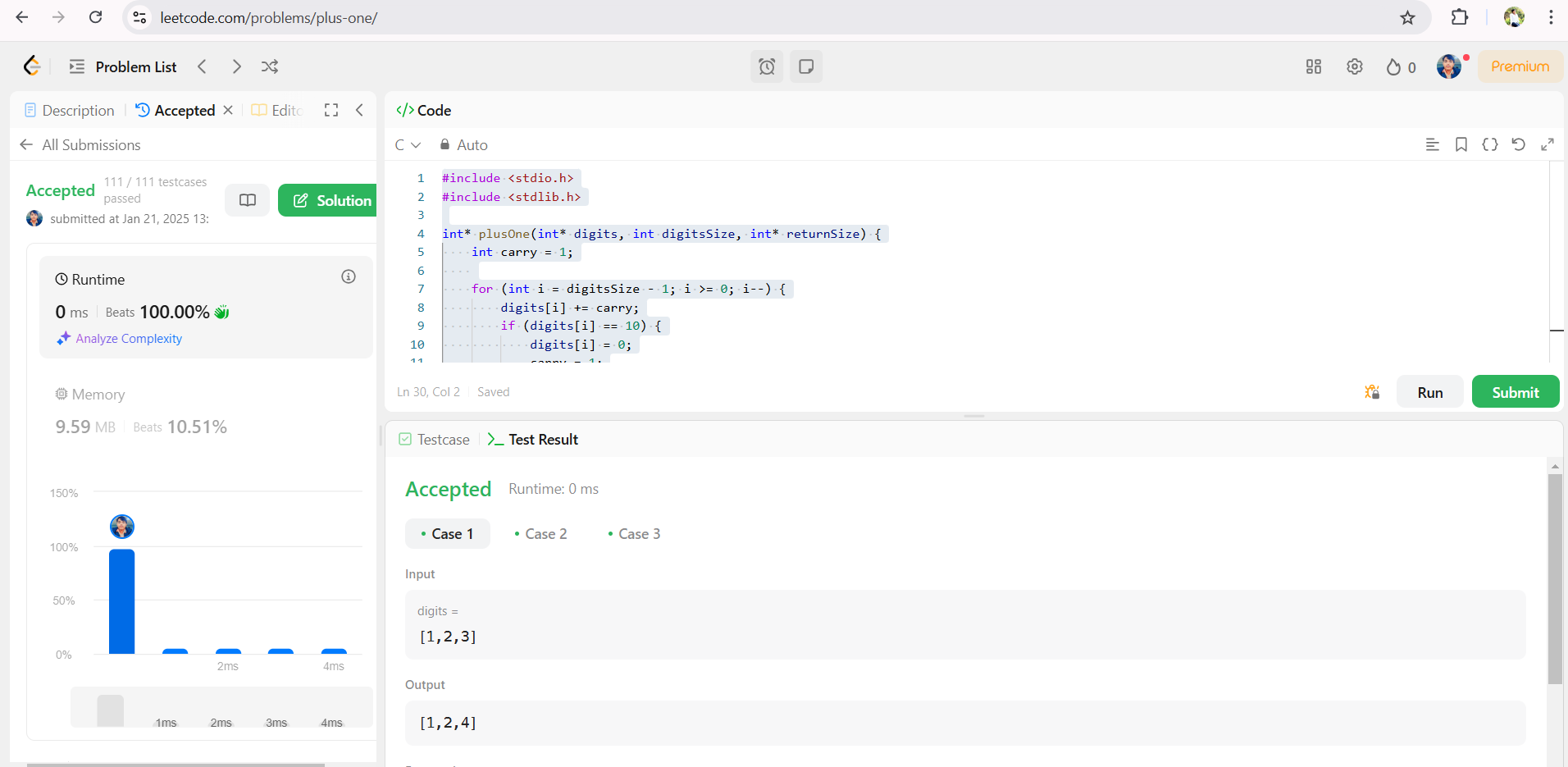
    } else {

        \*returnSize = digitsSize;

        return digits;

    }

}



**8)MISSING AND DUPLICATE NUMBER IN ARRAY**

int findDuplicate(int\* nums, int numsSize) {

    int slow = nums[0], fast = nums[0];

    do {

        slow = nums[slow];

        fast = nums[nums[fast]];

    } while (slow != fast);

    slow = nums[0];

    while (slow != fast) {

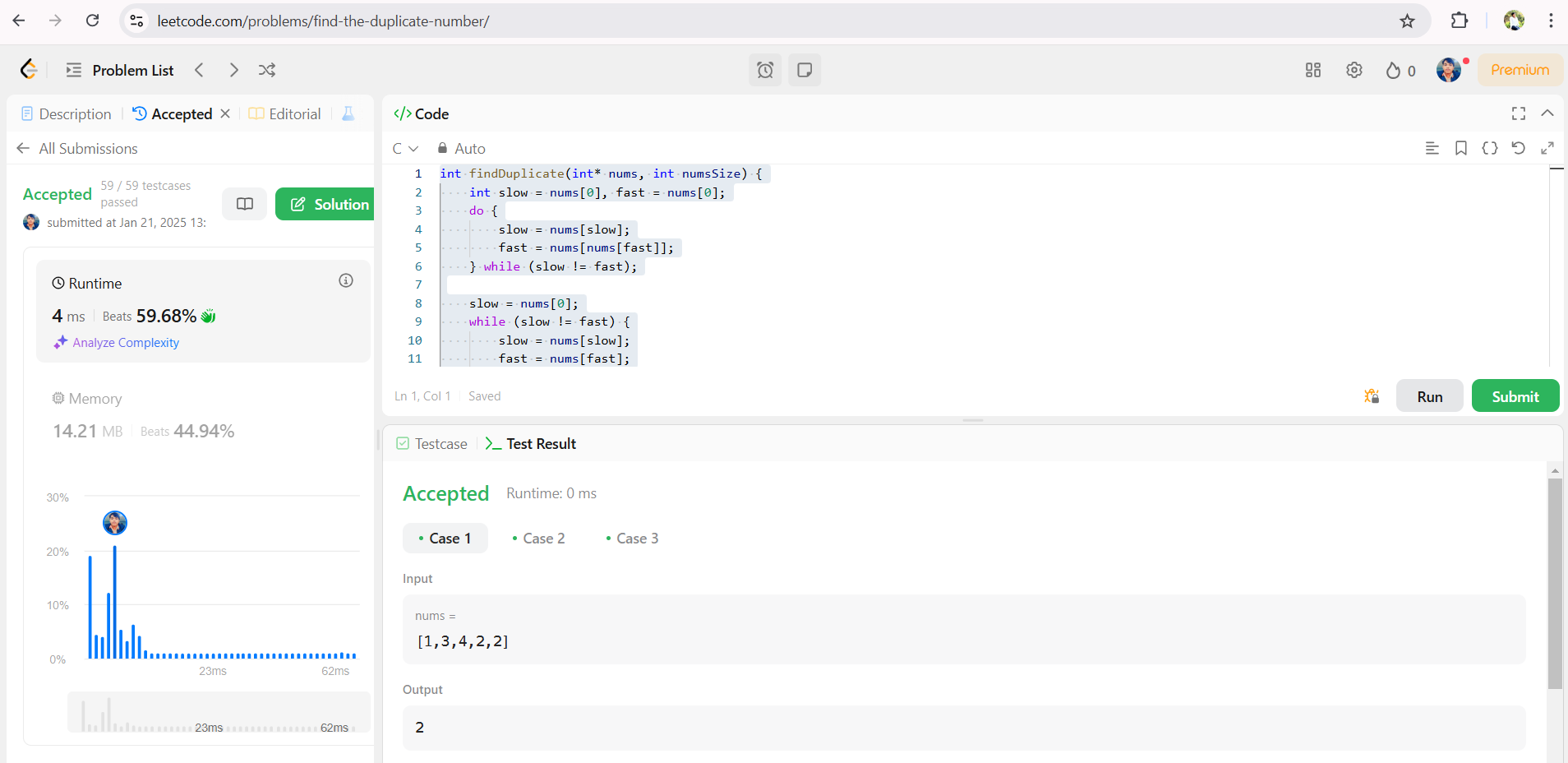
        slow = nums[slow];

        fast = nums[fast];

    }

    return slow;

}

****

**9)NUMBER N IS HAPPY**

#include <stdbool.h>

int getSumOfSquares(int num);

bool isHappy(int n) {

    int slow = n, fast = n;

    do {

        slow = getSumOfSquares(slow);

        fast = getSumOfSquares(getSumOfSquares(fast));

    } while (slow != fast);

    return slow == 1;

}

int getSumOfSquares(int num) {

    int sum = 0;

    while (num > 0) {

        int digit = num % 10;

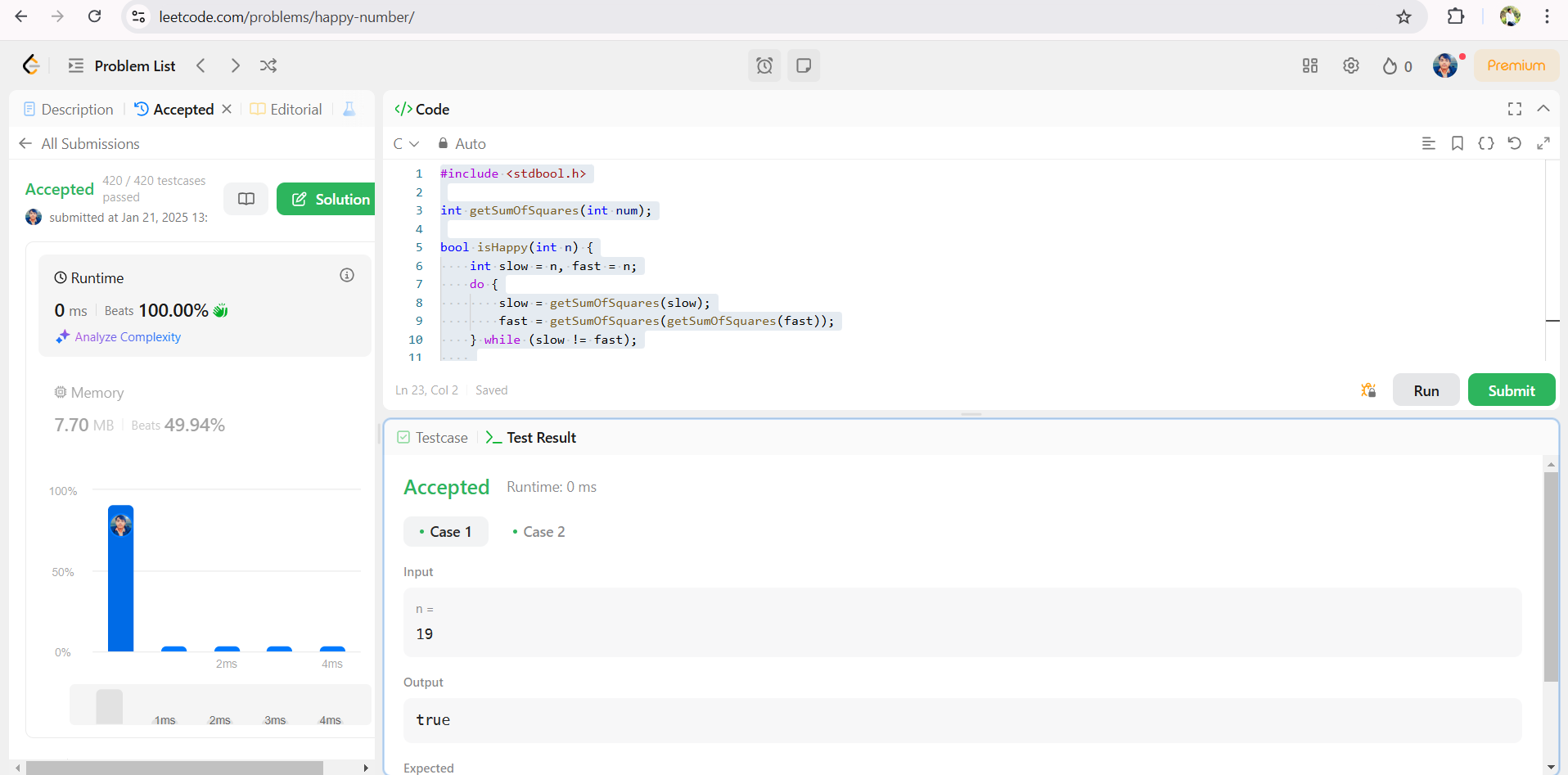
        sum += digit \* digit;

        num /= 10;

    }

    return sum;

}

****

**10)BINARY TREE-HEIGHT BALANCE**

#include <stdbool.h>

#include <stdlib.h>

int height(struct TreeNode\* root) {

    if (root == NULL) return 0;

    int leftHeight = height(root->left);

    if (leftHeight == -1) return -1;

    int rightHeight = height(root->right);

    if (rightHeight == -1) return -1;

    if (abs(leftHeight - rightHeight) > 1) return -1;

    return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);

}

bool isBalanced(struct TreeNode\* root) {

    return height(root) != -1;

}

